A General and Parallel Platform for Mining Co-Movement Patterns over Large-scale Trajectories

ABSTRACT

With the advances of modern positioning technologies, tremendous trajectory data are nowadays become widely available. Discovering insightful information out of such large-scale trajectories is important. Previous researches have identified several interesting co-movement patterns and showcased their usefulness. However, we observe two challenges in applying them over large-scale trajectories. First, there is lack of a uniform definition of co-movement pattern, which makes it cumbersome to design a tailored solution to feed different needs. Second, existing works are all centralized schemes which are failed to scale to hundreds of millions of trajectories.

Motivated by this, in this paper, we present our novel solution on detecting co-movement patterns over large-scale trajectories. First, we model a general co-movement pattern (GCMP) to capture co-moving behaviors of objects. GCMP is versatile in representing all existing patterns by introducing a *gap* constraints. With the gap, GCMP is also free from the so-called *loose-connection* anomaly which appears in existing methods. Second, we resort to MapReduce for a parallel GCMP mining solution. We propose a novel Star Partition and ApRiori Enumerator (SPARE) to facilitate efficient parallelism. In SPARE, we model the pairwised connections between objects as an Aggregate Graph and partition the graph into stars. Such a partition scheme guarantees no duplicate patterns are found from different partitions. For each star, SPARE employs an ApRiori Enumerator to systemically detect all valid GCMPs. As the traditional *monotonicity* property in Apriori no longer holds, we develop the temporal based *monotonicity* property to support efficient enumeration. We experiments our SPARE detector on three real datasets upto 200 million points using Apache Spark. The results show that SPARE achieves 90 times efficiency than centralized schemes and 10 times efficiency than a parallel baseline. Our experiments also demonstrates an almost linear scalability.

Patterns	Proximity	Consecutiveness	Time Complexity
flock [10]	disk-based	global	$O(\|\mathbb{O}\|\|\mathbb{T}\|(M + log(\|\mathbb{O}\|))$
convoy [7]	density-based	global	$O(\|\mathbb{O}\|^2 + \|\mathbb{O}\|\|\mathbb{T}\|)$
swarm [8]	density-based	-	$O(2^{\|\mathbb{O}\|}\|\mathbb{O}\|\|\mathbb{T}\|)$
group [6]	disk-based	local	$O(\|\mathbb{O}\|^2\|\mathbb{T}\|)$
platoon [9]	density-based	local	$O(2^{\|\mathbb{O}\|}\|\mathbb{O}\|\|\mathbb{T}\|)$

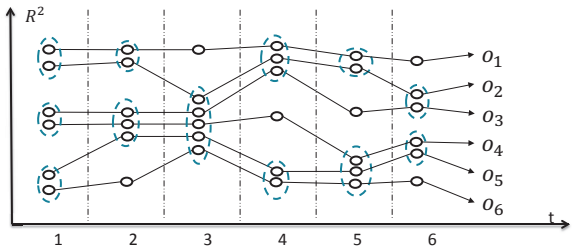
Table 1: Constraints and complexity of co-movement patterns. The time complexity indicates the performance in the worst case, where $|\mathbb{O}|$ is the total number of objects and $|\mathbb{T}|$ is the number of descritized timestamps.

1. INTRODUCTION

The prevalence of positioning devices has drastically boosted the scale and spectrum of trajectory collection to an unprecedented level. Tremendous amounts of trajectories, in the form of sequenced spatial-temporal records, are continually generated from animal telemetry chips, vehicle GPSs and wearable devices. Data analysis on large-scale trajectories benefits a wide range of applications and services, including traffic planning [1], animal analysis [2], and social recommendations [3], to name just a few.

A crucial task of data analysis on top of trajectories is to discover co-moving patterns. A *co-movement* pattern [4] refers to a group of objects traveling together for a certain period of time and the group is normally determined by spatial proximity. A pattern is prominent if the size of the group exceeds M and the length of the duration exceeds K, where M and K are parameters specified by users. Rooted from such basic definition and driven by different mining applications, there are a bunch of variants of co-movement patterns that have been developed with more advanced constraints.

Table 1 summarizes several popular co-moving pattern s with different constraints in the attributes of clustering in spatial proximity, consecutiveness in temporal duration and computational complexity. In particular, the *flock* [5] and the *group* [6] patterns require all the objects in a group to be enclosed by a disk with radius r; whereas the *convoy* [7], the *swarm* [8] and the *platoon* [9] patterns resort to density-based spatial clustering. In the temporal dimension, the *flock* [5] and the *convoy* [7] require all the timestamps of each detected spatial group to be consecutive, which is referred to as *global consecutiveness*; whereas the *swarm* [8] does not impose any restriction. The *group* [6] and the *platoon* [9] adopt a compromised manner by allowing arbitrary gaps between the consecutive segments, which is called *local consecutiveness*. They introduce a parameter L to control the minimum length of each local consecutive segment.



Flock(M=2, K=3)	$\langle o_3, o_4$: 1,2,3$\rangle$ $\langle o_5, o_6$: 3,4,5$\rangle$
Convoy(M=2, K=3)	$\langle o_3, o_4$: 1,2,3$\rangle$ $\langle o_5, o_6$: 3,4,5$\rangle$
Group(M=2, L=2)	$\langle o_1, o_2$: 1,2,4,5$\rangle$ $\langle o_3, o_4, o_5$: 2,3$\rangle$ $\langle o_5, o_6$: 3,4,5$\rangle$
Platoon (M=2,K=3,L=2)	$\langle o_1, o_2$: 1,2,4,5$\rangle$ $\langle o_3, o_4$: 1,2,3$\rangle$ $\langle o_4, o_5$: 2,3,5,6$\rangle$ $\langle o_5, o_6$: 3,4,5$\rangle$
Swarm (M=2,K=3)	$\langle o_1, o_2$: 1,2,4,5$\rangle$ $\langle o_3, o_4$: 1,2,3$\rangle$ $\langle o_4, o_5$: 2,3,5,6$\rangle$ $\langle o_5, o_6$: 1,3,4,5$\rangle$ $\langle o_2, o_3$: 3,4,6$\rangle$

Figure 1: Trajectories and co-movement patterns; The example consists of six trajectories across six snapshots. Objects in spatial clusters are enclosed by dotted circles. M is the minimum cluster cardinality; K denotes the minimum number of snapshots for the occurrence of a spatial cluster; and L denotes the minimum length for local consecutiveness.

Figure 1 is an example to demonstrate the concepts of various co-movement patterns. The trajectory database consists of six moving objects and the temporal dimension is discretized into six snapshots. In each snapshot, we treat the clustering methods as a black-box and assume that they generate the same clusters. Objects in proximity are grouped in the dotted circles. As aforementioned, there are three parameters to determine the co-movement patterns and the default settings in this example are $M = 2$, $K = 3$ and $L = 2$. Both the *flock* and the *convoy* require the spatial clusters to last for at least K consecutive timestamps. Hence,$\langle o_3, o_4 : 1, 2, 3\rangle$ and $\langle o_5, o_6 : 3, 4, 5\rangle$ remains the only two candidates matching the patterns. The *swarm* relaxes the pattern matching by discarding the temporal consecutiveness constraint. Thus, it generates many more candidates than the *flock* and the *convoy*. The *group* and the *platoon* add another constraint on local consecutiveness to retain meaningful patterns. For instance, $\langle o_1, o_2 : 1, 2, 4, 5\rangle$ is a pattern matching local consecutiveness because timestamps $(1, 2)$ and $(4, 5)$ are two segments with length no smaller than $L = 2$. The difference between the *group* and the *platoon* is that the *platoon* has an additional parameter K to specify the minimum number of snapshots for the spatial clusters. This explains why $\langle o_3, o_4, o_5 : 2, 3\rangle$ is a *group* pattern but not a *platoon* pattern.

As can be seen, there are various co-movement patterns requested by different applications and it is cumbersome to design a tailored solution for each type. In addition, despite the generality of the *platoon* (i.e., it can be reduced to other types of patterns via proper parameter settings), it suffers from the so-called *loose-connection* anomaly. We use two objects o_1 and o_2 in Figure 2 as an example to illustrate the scenario. These two objects form a *platoon* pattern in timestamps $(1, 2, 3, 102, 103, 104)$. However, the two consecutive segments are 98 timestamps apart, resulting in a false positive co-movement pattern. In reality, such an anomaly may be caused by the periodic movements of unrelated objects, such as vehicles stopping at the same petrol station or animals pausing at the same water source. Unfortunately, none of the existing patterns have directly addressed this anomaly.

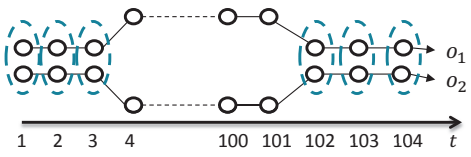
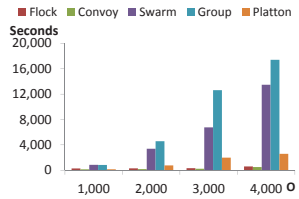
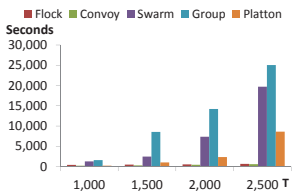


Figure 2: *Loose-connection* anomaly. Even though $\langle o_1, o_2 : 1, 2, 3, 102, 103, 104\rangle$ is considered as a valid *platoon* pattern, it is highly probable that these two objects are not related as the two consecutive segments are 98 timestamps apart.

The other issue with existing methods is that they are built on top of centralized indexes which may not be scalable. Table 1 shows their theoretical complexities in the worst cases and the largest real dataset ever evaluated in previous studies is up to million-scale points collected from hundreds of moving objects. In practice, the dataset is of much higher scale and the scalability of existing methods is left unknown. Thus, we conduct an experimental evaluation with 4000 objects moving for 2500 timestamps to examine the scalability. Results in Figure 3 show that their performances degrade dramatically as the dataset scales up. For instance, the detection time of *group* drops twenty times as the number of objects grows from *1k* to *4k*. Similarly, the performance of *swarm* drops over fifteen times as the number of snapshots grows from *1k* to *2.5k*. These observations imply that existing methods are not scalable to support large-scale trajectory databases.



(a) $1k$ timestamps



(b) $1k$ objects

Figure 3: Performance measures on existing co-movement patterns. A sampled Geolife data set is used with up two 2.4 million data points. Default parameters are $M = 15$ $K = 180$ $L = 30$.

Therefore, our primary contributions in this paper are to close these two gaps. First, we propose the *general co-movement pattern* (GCMP) which models various co-moment patterns in a unified way and can avoid the *loose-connection* anomaly. In GCMP, we introduce a new gap parameter G to pose a constraint on the temporal gap between two consecutive segments. By setting a feasible G, the loose-connection anomaly can be avoided. In addition, our GCMP is both general and expressive. It can be reduced to any of the previous patterns by customizing the parameters.

Second, we investigate deploying our GCMP detector on MapReduce platforms (such as Hadoop and Spark) to tackle the scalability issue. Our technical contributions are threefold. First, we replicate the snapshots in multiple data chunks to support efficient parallel processing. Second, we devise a novel *Star Partition and ApRiori Enumerator* (SPARE) framework as a fine-granularity partitioning strategy to achieve workload balance. For each star, an Apriori Enumerator is

adopted to mine the co-movement patterns. Third, we leverage the *temporal monotonicity* property of of GCMP to design several optimization techniques including *sequence simplification*, *monontonicity pruning* and *forward closure check* to further reduce the number of candidates enumerated.

We conduct a set of extensive experiments on three large-scaled real datasets with hundreds of millions temporal points. The results show that both our parallel scheme efficiently supports GCMP mining in large datasets. In particular, with near 200 million trajectory points, SPARE runs in 15 minutes using 162 cores. Whereas centralized solutions take near 7 hours for 1 million trajectory points. Moreover, our optimized SPARE methods achieves upto 10 times efficiency as compared to the baseline algorithm with almost linear scalability.

The rest of our paper is organized as follows: Section 2 summarizes the relevant literature on trajectory pattern mining. Section 3 states the problem definition of our general co-movement pattern mining. Section 4 provides a baseline solution. An advanced solution named *star partition and mining* is presented in Section 5. Section 6 conducts extensive experiments to verify the efficiency of our system. Finally Section 7 concludes the paper.

2. RELATED WORKS

The *co-movement patterns* in literature consist of five members, namely *group* [6], *flock* [10], *convoy* [7], *swarm* [8] and *platoon* [9]. We have demonstrated the semantics of these patterns in Table 1 and Figure 1. In this section, we focus on comparing the techniques used in these works. For more trajectory patterns other than *co-movement patterns*, interested readers may move to [11] for a comprehensive survey.

2.1 Flock and Convoy

The difference between *flock* and *convoy* lies in the object clustering methods. In *flock* objects are clustered based on their distance. Specifically, the objects in the same cluster needs to have a pair-wised distance less than *min_dist*. This essentially requires the objects to be within a disk-region of delimiter less than *min_dist*. In contrast, *convoy* cluster the objects using density-based clustering [21]. Technically, *flock* utilizes a m^{th}-order Voronoi diagram [13] to detect whether a subset of object with size greater than m stays in a disk-region. *Convoy* employs a trajectory simplification [14] technique to boost pairwise distance computations in the density-based clustering. After clustering, both *flock* and *convoy* use a line-sweep method to scan each snapshots. During the scan, the object group appears in consecutive timestamps is detected. Meanwhile, the object groups that do not match the consecutive constraint are pruned. However, such a method faces high complexity issues when supporting other patterns. For instance, in *swarm*, the candidate set during the line-sweep grows exponentially, and many candidates can only be pruned after the entire snapshots are scanned.

2.2 Group, Swarm and Platoon

Different from *flock* and *convoy*, all the *group*,*swarm* and *platoon* patterns have more constraints on the pattern duration. Therefore, their techniques of mining are of the same skeleton. The main idea of mining is to grow object set from an empty set in a depth-first manner. During the growth, various pruning techniques are provided to prune unnecessary branches. *Group* pattern uses the Apriori property among patterns to facilitate the pruning. *Swarm* adapts two more pruning rules called backward pruning and forward pruning. *Platoon* further adapts a prefix table structure to guide the depth-first search. As shown by Li et.al. [9], *platoon* outperforms other two methods in efficiency. However, the three patterns are not able to directly discover the general co-movement pattern. Furthermore, their pruning rules heavily rely on the depth-first search nature, which lost its efficiency in the parallel scenario.

3. DEFINITIONS

Let $\mathbb{O} = \{o_1, o_2, ..., o_n\}$ be the set of objects and $\mathbb{T} = (1, 2, ..., N)$ be the discretized temporal dimension. A time sequence T is defined as a ordered subset of $\mathbb{T}$. Given two time sequences T_1 and T_2, we define a bunch of commonly-used operators in this paper in Table 2.

Operator	Definition
$T[i]$	the i-th element in the sequence T
$\|T\|$	the number of elements in T
$\max(T)$	the maximum element in T
$\min(T)$	the minimum element in T
$\text{range}(T)$	the range of T, i.e., $\max(T) - \min(T) + 1$
$T[i:j]$	subsequence of T from $T[i]$ to $T[j]$ (inclusive)
$T_1 \subseteq T_2$	$\forall T_1[x] \in T_1$, we have $T_1[x] \in T_2$.
$T_3 = T_1 \cup T_2$	$\forall T_3[x] \in T_3$, we have $T_3[x] \in T_1$ or $T_3[x] \in T_2$
$T_3 = T_1 \cap T_2$	$\forall T_3[x] \in T_3$, we have $T_3[x] \in T_1$ and $T_3[x] \in T_2$

Table 2: Operators on time sequence.

We say a sequence T is consecutive if $\forall i \in (1, ..., |T| - 1), T[i+1] = T[i]+1$. We refer each consecutive subsequence of T as a *segment*. It is obvious that any time sequence T can be decomposed into segments and we say T is *L-consecutive* [9] if the length of every segment is no smaller than L. As illustrated in Figure 2, patterns adopting the notion of L-consecutiveness (e.g., *platoon* and *group*) still suffer from the *loose connection* problem. To avoid such an anomaly without losing pattern generality, we introduce a parameter G to control the gaps between timestamps in a pattern. Formally, a G-connected time sequence is defined as follows:

Definition 1 (G-connected)**.** *A time sequence T is G-connected if the gap between any of its neighboring timestamps is no greater than G. That is $\forall i \in (1, ..., |T| - 1), T[i + 1] - T[i] \leq G$.*

We take $T = (1, 2, 3, 5, 6)$ as an example, which can be decomposed into two segments $(1, 2, 3)$ and $(5, 6)$. T is not 3-consecutive since the length of $(5, 6)$ is 2. Thus, it is safe to say either T is 1-consecutive or 2-consecutive. On the other hand, T is 2-connected since the maximum gap between its neighboring time stamps is $5 - 3 = 2$. It is worth noting that T is not 1-connected because the gap between $T[3]$ and $T[4]$ is 2 (i.e., 5-3=2).

Given a trajectory database discretized into snapshots, we can conduct a clustering method, either disk-based or density-based, to identify groups with spatial proximity. Let T be the set of timestamps in which a group of objects O are clustered. We are ready to define a more general co-movement pattern:

Definition 2 (General Co-Movement Pattern)**.** *A general co-movement pattern finds a set of objects O satisfying the*

following five constraints: (1) closeness: the objects in O belong to the same cluster in the timestamps of T; (2) significance: $|O| \geq M$; (3) duration: $|T| \geq K$; (4) consecutiveness: T is L-consecutive; and (5) connection: T is G-connected.

There are four parameters in our general co-movement pattern, including object constraint M and temporal constraints K, L, G. By customizing these parameters, our pattern can express other patterns proposed in previous literature, as illustrated in Table 3. In particular, by setting $G = |\mathbb{T}|$, we achieve the *platoon* pattern. By setting $G = |\mathbb{T}|, L = 1$, we achieve the *swarm* pattern. By setting $G = |\mathbb{T}|$, $M = 2$, $K = 1$, we gain the *group* pattern. Finally by setting $G = 1$, we achieve the *convoy* and *flock* patterns. In addition to the flexibility of representing other existing patterns, our GCMP is able to avoid the *loose connection* anomaly by tuning the parameter G. It is notable that GCMP cannot be modeled by existing patterns.

Pattern	M	K	L	G	**Clustering**
Group	2	1	2	$\|\mathbb{T}\|$	Disk-based
Flock	·	·	K	1	Disk-based
Convoy	·	·	K	1	Density-based
Swarm	·	·	1	$\|\mathbb{T}\|$	Density-based
Platoon	·	·	·	$\|\mathbb{T}\|$	Density-based

Table 3: Expressing other patterns using GCMP. · indicates a user specified value. M represents the object *size* constraint. K represents the *duration* constraint. L represents the *consecutiveness* constraint. G represents the *connection* constraint.

Our definition of GCMP is independent of the clustering method. Users can apply different clustering methods to facilitate different application needs. We currently expose both disc-region based clustering and DBSCAN as options to the users. In summary, the goal of this paper is to present a parallel solution for discovering all the valid GCMP from large-scale trajectory datasets. Before we move on to the algorithmic part, we list the notations that are used in the following sections.

Symbol	Meaning
S_t	snapshot of objects at time t
M	object size constraint
K	duration constraint
L	consecutiveness constraint
G	connection constraint
$P = \langle O : T \rangle$	pattern with object set O, time sequence T
η	replication factor in the TRPM framework
$C_t(o)$	the cluster of object o at time t
S_t	the set of clusters at time t
λ_t	the partition with snapshots $S_t, .., S_{t+\eta-1}$
Sr_i	the star partition for object i

Table 4: Symbols and notions used

4. BASELINE: TEMPORAL REPLICATION AND PARALLEL MINING

In this section, we propose a baseline solution that resorts to MapReduce (MR) as a general, parallel and scalable paradigm for GCMP pattern mining. The framework, named *temporal replication and parallel mining* (TRPM), is illustrated in Figure 4. There are two cycles of map-reduce jobs connected in a pipeline manner. The first cycle deals with spatial clustering in each snapshot, which can be seen as a preprocessing step for the subsequent pattern mining. In particular, the timestamp is treated as the key in the map phase and objects within the same snapshot are clustered (DBSCAN or disk-based clustering) in the reduce phase. Finally, the reducers output clusters of objects in each snapshot, represented by a list of key-value pairs $\langle t, S_t \rangle$, where t is the timestamp and S_t is a set of clustered objects at snapshot t.

Our focus in this paper is the second map-reduce cycle of parallel mining, which essentially consists of two key questions to solve. The first is how to employ effective data partitioning such that the mining can be conducted independently; and the second is how to efficiently mine the valid patterns within each partition.

It is obvious that we cannot simply split the trajectory database into disjoint partitions because a GCMP pattern requires L-consecutiveness and the corresponding segments may cross multiple partitions. Our strategy is to use data replication to enable parallel mining. Each snapshot will replicate its clusters to $\eta - 1$ preceding snapshots. In other words, the partition for the snapshot S_t contains clusters in S_t, $S_{t+1} \ldots, S_{t+\eta-1}$. Determining a proper η is critical in ensuring the correctness and efficiency of TRPM. If η is too small, certain cross-partition patterns may be missed. If η is set too large, expensive network communication and CPU processing costs would be incurred in the map and reduce phases respectively. Our objective is to find an η that is not large but can guarantee correctness.

In our implementation, we set $\eta = (\lceil \frac{K}{L} \rceil - 1) * (G-1) + K + L - 1$. Intuitively, K timestamps generates at most $\lceil \frac{K}{L} \rceil - 1$ gaps as the length of each L-consecutive segment is at least L. Since the gap size is at most $G - 1$, $(\lceil \frac{K}{L} \rceil - 1) * (G - 1)$ is the upper bound of timestamps allocated to gaps. The remaining part $K + L - 1$ is used to capture the upper bound allocated for the L-consecutive segments. We formally prove that η can guarantee correctness.

Theorem 1. *$\eta = (\lceil \frac{K}{L} \rceil - 1) * (G-1) + K + L - 1$ guarantees that no valid pattern is missing.*

Proof. Given a valid pattern P, we can always find at least one valid subsequence that is also valid. Let T' denote the valid subsequence with the minimum length. In the worst case, $T' = P.T$. We define $\text{range}(T) = \max(T) - \min(T) + 1$ and prove the theorem by showing that $\text{range}(T') \leq \eta$. Since T' can be written as a sequence of L-consecutive segments interleaved by gaps: $l_1, g_1, \ldots, l_{n-1}, g_{n-1}, l_n$ $(n \geq 1)$, where l_i is a segment and g_i is a gap. Then, $\text{range}(T')$ is calculated as $\Sigma_{i=1}^{i=n} |l_i| + \Sigma_{i=1}^{i=n-1} |g_i|$. Since T' is valid, then $\Sigma_{i=1}^{i=n} |l_i| \geq K$. As T' is minimum, if we remove the last l_n, the resulting sequence should not be valid. Let $K' = \Sigma_{i=1}^{i=n-1} |l_i|$, which is the size of the first $(n-1)$ segments of T'. Then, $K' \leq K - 1$. Note that every $|l_i| \geq L$, thus $n \leq \lceil \frac{K'}{L} \rceil \leq \lceil \frac{K}{L} \rceil$. By using the fact that every $|g_i| \leq G - 1$, we achieve $\Sigma_{i=1}^{i=n-1} |g_i| \leq (n-1)(G-1) \leq (\lceil \frac{K}{L} \rceil - 1)(G-1)$. Next, we consider the difference between K and K', denoted by $\Delta = K - K'$. To ensure T''s validity, l_n must equal to $\min(L, \Delta)$. Then, $\Sigma_{i=1}^{i=n} |l_i| = K' + l_n = K - \Delta + \min(L, \Delta) \leq K - 1 + L$. We finish showing $\text{range}(T') \leq \eta$. Therefore, for any valid sequence T, it exists at least one valid subsequence

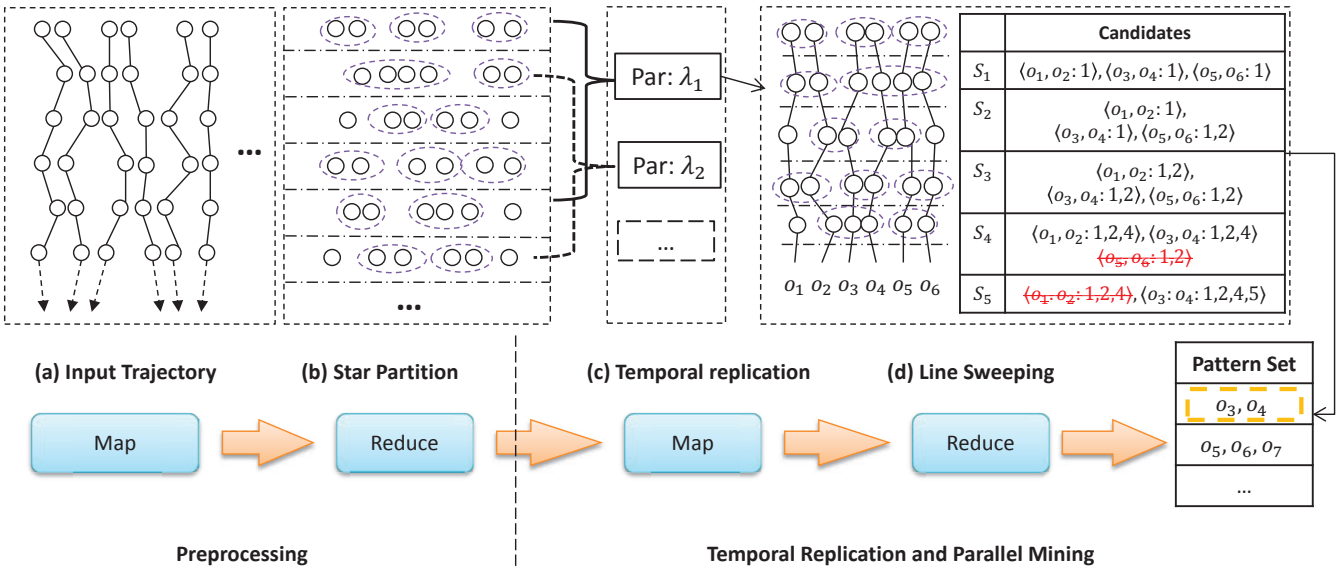


Figure 4: Work flow of Temporal Replication and Parallel Mining (TRPM). (a)(b) correspond to the first map-reduce cycle which clusters objects in each snapshot; (c)(d) correspond to the second map-reduce cycle, which uses TRPM to detect GCMP in parallel.

with range no greater than η and hence this pattern can be detected in a partition with η snapshots. □

Based on the above theorem, during TRPM, every consecutive η snapshots form a partition. In other words, each snapshot S_t corresponds to a partition $\lambda_t = \{S_t, ..., S_{t+\eta-1}\}$. Our next task is to design an efficient pattern mining strategy within each partition. We propose a line sweep algorithm to sequentially scan the η replicated snapshots in a partition and employ effective candidate pattern enumeration.

Details of the algorithm are presented in Algorithm 1. We keep a candidate set C (Line 1) during the sweeping process. It is initialized as the candidate clusters with size no smaller than M in the first snapshot. Then, we sequentially scan each snapshot (Lines 7-27) and generate new candidates by extending the original ones in C. In particular, we join candidates in C with all the clusters in S_j to form new candidates (Lines 9-16). After sweeping all the snapshots, all the valid patterns are stored in C(Line 28). It is worth noting that C continues to grow during the whole sweeping process. We can use three pruning rules to early remove false candidates from C. Since there is a partition λ_t for each S_t, only patterns that start from timestamp t need to be discovered. Therefore, those patterns that does not appear in the S_t are false candidates. Particularly, our three pruning rules are as follows: First, when sweeping snapshot S_j, new candidates with objects set smaller than M are pruned (Line 14). Second, after joined with all clusters in S_j, candidates in C with the maximum timestamp no greater than $j - G$ are pruned (Lines 18-21). Third, candidates in C with the size of first segment smaller than L are pruned (Lines 22-24). With the three pruning rules, the size of C could be significantly reduced.

The complete picture of temporal replication and parallel mining is summarized in Algorithm 2. We illustrate the workflow of TRPM method using Figure 4 (c)(d) with pattern parameters $M = 2, K = 3, L = 2, G = 2$. By Theorem 1, η is calculated as $(\lceil \frac{K}{L} \rceil - 1) * (G - 1) + 2K - 2 = 5$. Therefore, in Figure 4 (c), every 5 consecutive snapshots are combined into a partition in the map phase. In Figure 4 (d), a line sweep method is illustrated for partition λ_1. Let C_i be the candidate set during sweeping snapshot S_i. Initially, C_1 contains patterns with object sets in snapshot S_1. As we sweep the snapshots, the patterns in C_i grow. At snapshot S_4, the candidate $\langle o_5, o_6\rangle$ is removed. This is because the gap between its latest timestamp (i.e., 2) and the next scanning timestamp (i.e., 5) is 3, which violates the G-connected constraint. Next, at snapshot S_5, the candidate $\langle o_1, o_2\rangle$ is removed. This is because its local consecutive segment (4) has only 1 element, which violates the L-consecutive constraint. Finally, $\langle o_3, o_4\rangle$ is the valid pattern and is returned. Note that in this example, $\eta = 5$ is the minimum setting that can guarantee correctness. If η is set to 4, the pattern $\langle o_3, o_4\rangle$ would be missed.

5. SPARE: STAR PARTITIONING AND APRIORI ENUMERATOR

The aforementioned replicate partitioning is based on the temporal dimension which suffers from two drawbacks. First, the replication relies on η which could be large. Second, the same valid pattern may be discovered from different partitions which results in redundant works. To resolve the limitations caused by the replicate partitioning, we propose a new Star Partitioning and ApRiori Enumerator, named SPARE, to replace the second cycle of map-reduce jobs in Figure 4. Our new parallel mining framework is shown in Figure 5. Its input is the set of clusters generated in each snapshot and the output contains all the valid GCMP patterns. In the following, we explain the two major components: star partitioning and apriori enumerator.

5.1 Star Partitioning

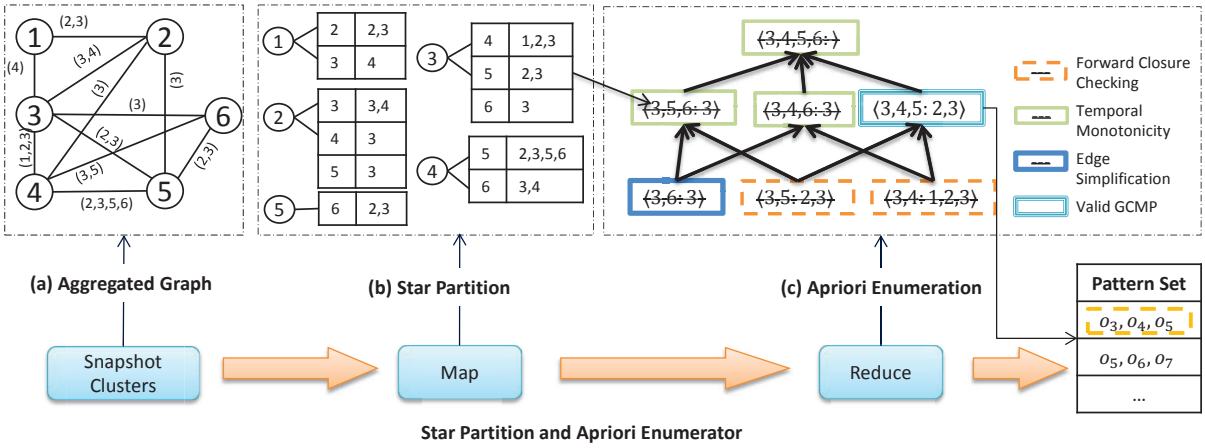


Figure 5: Star partition and ApRiori Enumerator (SPARE). (a) Aggregate graph from Figure 1. (b) Five stars are generated, star IDs are circled, the vertexes and inverted lists are in the connected tables. (c) Apriori Enumerator with various pruning techniques.

Let G_t be a graph for snapshot S_t, in which each node is a moving object and two objects are connected if they appear in the same cluster. It is obvious that G_t consists of a set of small cliques. Based on G_t, we define an aggregated graph G_A to summarize the cluster relationship among all the snapshots. In G_A, two objects form an edge if they are connected in any G_ts. Furthermore, we attach an inverted list for each edge, storing the associated timestamps in which the two objects are connected. An example of G_A, built on the trajectory database in Figure 1, is shown in Figure 5 (a). As long as two objects are clustered in any timestamps, they are connected in G_A. The object pair $\langle o_1, o_2 \rangle$ appears in two clusters at timestamps 2 and 3 and is thus associated with an inverted list $(2, 3)$.

We use *star* as the data structure to capture the pair relationships. To avoid duplication, as G_t is an undirected graph and an edge may appear in multiple stars, we enforce a global ordering among the objects and propose a concept named *directed star*.

Definition 3 (Directed Star)**.** *Given a vertex with global id s, its directed star Sr_s is defined as the set of neighboring vertices with global id $t > s$. We call s the star ID.*

With the global ordering, we can guarantee that each edge is contained in a unique star partition. Given the aggregated graph G_A in Figure 5 (a), we enumerate all the possible directed stars in Figure 5 (b). These stars are emitted from mappers to different reducers. The key is the star ID and the value is the neighbors in the star as well as the associated inverted lists. The reducer will then call the Apriori-based algorithm to enumerate all the valid GCMP patterns.

Before we introduce the Apriori enumerator, we are interested to examine the issue of global ordering on the moving objects. This is because assigning different IDs to the objects will result in different star partitioning results, which will eventually affect the workload balance among the map-reduce jobs. The job incurring performance bottleneck is often known as *straggler* [15, 16, 17]. In the context of star partitioning, a straggler refers to the job assigned with the maximum star partition. We use Γ to denote the size of a partition and Γ is set to the number of edges in a directed star[1]. It is straightforward that a star partitioning with small Γ is preferred. For example, Figure 6 gives two star partitioning results under different vertex ordering on the same graph. The top one has $\Gamma = 5$ while the bottom one has $\Gamma = 3$. Obviously, the bottom one with smaller Γ is much more balanced.

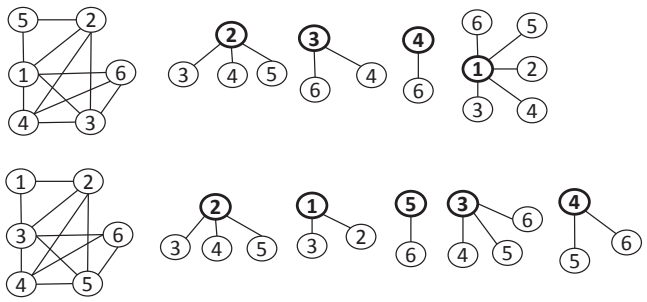


Figure 6: Examples of star partitioning with different vertex ordering.

Although it is very challenging to find the optimal vertex id ordering from the $n!$ possibilities, we observe that a random order can actually achieve satisfactory performance based on the following theorem.

Theorem 2. *Let Γ^* be the value derived from the optimal vertex ordering and Γ be value derived from a random vertex ordering. With probability $1 - 1/n$, we have $\Gamma = \Gamma^* + O(\sqrt{n \log n})$.*

Proof. In Appendix A.1. □

If G_A is a dense graph, we can get a tighter bound for $(\Gamma - \Gamma^*)$.

Theorem 3. *Let d be the average degree in G_A. If $d \geq \sqrt{12 \log n}$, with high probability $1 - 1/n$, $\Gamma = \Gamma^* + O(\sqrt{d \log n})$.*

Proof. In Appendix A.1. □

[1]A star is essentially a tree structure and the number of nodes equals the number of edges minus one.

Algorithm 1 Line Sweep Mining

Require: $\lambda_t = \{S_t, ..., S_{t+\eta-1}\}$
$C \leftarrow \{\}$ ▷ Candidate set
for all clusters s in snapshot S_t **do**
  if $|s| \geq M$ **then**
    $C \leftarrow C \cup \{\langle s, t\rangle\}$
  end if
end for
for all $S_j \in \{S_{t+1}, \ldots, S_{t+\eta-1}\}$ **do**
  $N \leftarrow \{\}$
  for all $(c, s) \in C \times S_j$ **do**
    $c' \leftarrow \langle c.O \cap s.O, c.T \cup \{j\}\rangle$
    if $c'.T$ is valid **then**
      output c'
    else if $|c'.O| \geq M$ **then**
      $N \leftarrow N \cup \{c'\}$
    end if
  end for
  for all $c \in C$ **do**
    if $j - \max(c.T) \geq G$ **then**
      $C \leftarrow C - \{c\}$
      output c, if c is a valid pattern
    end if
    if c's first segment is less than L **then**
      $C \leftarrow C - \{c\}$
    end if
  end for
  $C \leftarrow C \cup N$
end for
output valid patterns in C

Algorithm 2 Temporal Replication and Parallel Mining

Require: list of $\langle t, S_t\rangle$ pairs
$\eta \leftarrow (\lceil \frac{K}{L} \rceil - 1) * (G - 1) + K + L - 1$
—Map Phase—
for all snapshots S_t **do**
  for all $i \in 1...\eta - 1$ **do**
    emit key-value pair $\langle \max(t - i, 0), S_t\rangle$
  end for
end for
—Partition and Shuffle Phase—
for all key-value pairs $\langle t, S\rangle$ pair **do**
  group-by t and emit a key-value pair $\langle t, \lambda_t\rangle$, where $\lambda_t = \{S_t, S_{t+1}, ..S_{t+\eta-1}\}$
end for
—Reduce Phase—
for all key-value pairs $\langle t, \lambda_t\rangle$ **do**
  call line sweep algorithm for partition λ_t
end for

Hence, we can simply use object id to determine the vertex ordering in our implementation.

5.2 Apriori Enumerator

Intuitively, given a GCMP pattern with an object set $\{o_1, o_2, \ldots, o_m\}$, all the pairs of $\langle o_i, o_j\rangle$ with $1 \leq i < j \leq m$ must be connected in the associated temporal graphs $\{G_t\}$. This inspires us to leverage the classic Apriori algorithm to enumerate all the valid GCMP patterns starting from pairs of objects. However, we observe that the monotonicity property does not hold between an object set and its supersets.

Example 1. *In this example, we show that if an object set is not a valid pattern, we cannot prune all its super sets. Consider two candidates* $P_1 = \langle o_1, o_2 : 1, 2, 3, 6\rangle$ *and* $P_2 = \langle o_1, o_3 : 1, 2, 3, 7\rangle$. *Let* $L = 2, K = 3$ *and* $G = 2$. *Both candidates are not valid patterns because the constraint on L is not satisfied. However, when considering their object superset* $\langle o_1, o_2, o_3\rangle$, *we can infer that their co-clustering timestamps are in* $(1, 2, 3)$. *This is a valid pattern conforming to the constraints of* L, K, G. *Thus, we need a new type of monotonicity to facilitate pruning.*

5.2.1 Monotonicity

To ensure the monotonicity, we first introduce a procedure named *star simplification*, to reduce the number of edges as well as unnecessary timestamps in the inverted lists. For instance, if the size of the inverted list for an edge e is smaller than K, then the edge can be safely removed because the number of timestamps in which its supersets are clustered must also be smaller than K. To generalize the idea, we propose three concepts named *maximal G-connected subsequence*, *decomposable sequence* and *sequence simplification*.

Definition 4 (Maximal G-connected Subsequence)**.** *A sequence* T' *is said to be a maximal* G*-connected subsequence of* T *if (1)* T' *is the subsequence of* T, *i.e.,* $\exists i \leq j, T' = T(i, \ldots, j)$ *, (2)* T' *is* G*-connected, and (3) there exists no other subsequence* T'' *of* T *such that* T' *is the subsequence of* T'' *and* T'' *is* G*-connected.*

Example 2. *Suppose* $G = 2$ *and consider two sequences* $T_1 = (1, 2, 4, 5, 6, 9, 10, 11, 13)$ *and* $T_2 = (1, 2, 4, 5, 6, 8, 9)$. T_1 *has two maximal* 2*-connected subsequences:*$T_1^A = (1, 2, 4, 5, 6)$ *and* $T_1^B = (9, 10, 11, 13)$. *This is because the gap between* T_1^A *and* T_1^B *is* 3 *and it is impossible for the timestamps from* T_1^A *and* T_1^B *to form a new subsequence with* $G \leq 2$. *Since* T_2 *is* 2*-connected,* T_2 *has only one maximal* 2*-connected subsequence which is itself.*

The maximal G-connected subsequence has the following two properties:

Lemma 4. *Suppose* $\{T_1, T_2, \ldots, T_m\}$ *is the set of all maximal* G*-connected subsequences of* T, *we have (1)* $T_i \cap T_j = \emptyset$ *for* $i \neq j$ *and (2)* $T_1 \cup T_2 \cup \ldots \cup T_m = T$.

Proof. We assume $T_i \cap T_j \neq \emptyset$. Let $T_i = (T_i[1], T_i[2], \ldots, T_i[p])$ and $T_j = (T_j[1], T_j[2], \ldots, T_j[n])$. Suppose $T[x]$ is a timestamp occurring in both T_i and T_j. Let $T[y] = \min\{T_i[1], T_j[1]\}$, i.e., the minimum timestamp of $T_i[1]$ and $T_j[1]$ occurs at the y-th position of sequence T. Similarly, we assume $T[z] = \max\{T_i[p], T_j[n]\}$. Apparently, the two subsequences $T[y : x]$ and $T[x : z]$ are G-connected because T_i and T_j are both G-connected. Then, sequence $(T_y, \ldots, T_x, \ldots, T_z)$, the superset of T_i and T_j, is also G-connected. This contradicts with the assumptions that T_i and T_j are maximal G-connected subsequences.

To prove (2), we assume $\cup_i T_i$ does not cover all the timestamps in T. Then, we can find a subsequence $T' = T[x : x + t]$ such that $T[x - 1] \in T_a$ $(1 \leq a \leq m)$, $T[x + t + 1] \in T_b$ $(1 \leq b \leq m)$ and all the timestamps in T' is not included in any T_i. Let $g' = \min\{T[x] - T[x - 1], T[x + t + 1] - T[x + t]\}$. If $g' \leq G$, then it is easy to infer that T_a or T_b is not a maximal G-connected subsequence because we can combine

it with $T[x]$ or $T[x+t]$ to a form superset which is also G-connected. If $g' > G$, T' itself is a maximal G-connected subsequence which is missed in $\cup T_i$. Both cases lead to contradiction. □

Lemma 5. *If T_1 is a subset of T_2, then for any maximal G-connected subsequence T_1' of T_1, we can find a maximal G-connected subsequence T_2' of T_2 such that T_1' is a subset of T_2'.*

Proof. Since $T_1' \subseteq T_1 \subseteq T_2$, we know T_1' is a G-connected subsequence of T_2. Based on Lemma 4, we can find a maximal G-connected subsequence of T_2, denoted by T_2', such that $T_1' \cap T_2' \neq \emptyset$. If there exists a timestamp $T_1'[x]$ such that $T_1'[x] \notin T_2'$, similar to the proof of case (1) in Lemma 4, we can obtain a contradiction. Thus, all the timestamps in T_1' must occur in T_2'. □

Definition 5 (Decomposable Sequence)**.** *T is decomposable if for any of its maximal G-connected subsequence T', we have (1) T' is L-consecutive; and (2) $|T'| \geq K$.*

Example 3. *Let $L = 2, K = 4$ and we follow the above example. T_1 is not a decomposable sequence because one of its maximal 2-connected subsequence (i.e., T_1^B) is not 2-consecutive. In contrast, T_2 is a decomposable sequence because the sequence itself is the maximal 2-connected subsequence, which is also 2-consecutive and with size no smaller than than 4.*

Definition 6 (Sequence Simplification)**.** *Given a sequence T, the simplification procedure $\texttt{sim}(T) = g_{G,K} \cdot f_L(T)$ can be seen as a composite function with two steps:*

1. *f-step: remove segments of T that are not L-consecutive;*
2. *g-step: among the maximal G-connected subsequences of $f_L(T)$, remove those with size smaller than K.*

Example 4. *Take $T = (1, 2, 4, 5, 6, 9, 10, 11, 13)$ as an example for sequence simplification. Let $L = 2, K = 4$ and $G = 2$. In the f-step, T is reduced to $f_2(T) = (1, 2, 4, 5, 6, 9, 10, 11)$. The segment (13) is removed due to the constraint of $L = 2$. $f_2(T)$ has two maximal 2-consecutive subsequences: $(1, 2, 4, 5, 6)$ and $(9, 10, 11)$. Since $K = 4$, we will remove $(9, 10, 11)$ in the g-step. Finally, the output is $\texttt{sim}(T) = (1, 2, 4, 5, 6)$.*

It is possible that the simplified sequence $\texttt{sim}(T) = \emptyset$. For example, Let $T = (1, 2, 5, 6)$ and $L = 3$. All the segments will be removed in the f-step and the output is $\emptyset$. We define $\emptyset$ to be not decomposable. We provide an important property of the sequence simplification process as follows:

Lemma 6. *If sequence T is a superset of any decomposable sequence, then $\texttt{sim}(T) \neq \emptyset$.*

Proof. It is obvious that $\texttt{sim}(T)$ is a one-to-one function. Given an input sequence T, there is a unique $\texttt{sim}(T)$. Let T_p be a decomposable subset of T and we prove the lemma by showing that $\texttt{sim}(T)$ is a superset of T_p.

Suppose T_p can be decomposed into a set of maximal G-connected subsequences $T_p^1, \ldots, T_p^m$ $(m \geq 1)$. Since T_p is a subset of T, all the T_p^i are also subsets of T. By definition, each T_p^i is L-consecutive. Thus, in the f-step of $\texttt{sim}(T)$, none of T_p^i will be removed. In the g-step, based on Lemma 5, we know that each T_p^i has a superset in the maximal G-connected subsequences of $f_L(T)$. Since $|T_p^i| \geq K$, none of T_p^i will be removed in the g-step. Therefore, all the T_p^i will be retained after the simplification process and $\texttt{sim}(T) \neq \emptyset$. □

With Lemma 6, we are ready to define the *monotonicity* based on the simplified sequences to facilitate the pruning in the Apriori algorithm.

Theorem 7 (Monotonicity)**.** *Given a candidate pattern $P = \{O : T\}$, if $\texttt{sim}(P.T) = \emptyset$, then any pattern candidate P' with $P.O \subseteq P'.O$ can be pruned.*

Proof. We prove by contradiction. Suppose there exists a valid pattern P_2 such that $P_2.O \supseteq P.O$. It is obvious that $P_2.T \subseteq P.T$. Based on the Definition 2, the following conditions hold: (1) $P_2.T$ is G-connected. (2) $|P_2.T| \geq K$ and (3) $P_2.T$ is L-consecutive. Note that the entire $P_2.T$ is G-connected. Thus, $P_2.T$ itself is the only maximal G-connected subsequence. Based on conditions (1),(2),(3) and Definition 6, $P_2.T$ is decomposable. Then, based on Lemma 6, we know $\texttt{sim}(T) \neq \emptyset$ because $P_2.T \subseteq P.T$ and $P_2.T$ is decomposable. This leads to a contradiction with $\texttt{sim}(P.T) = \emptyset$. □

5.2.2 Apriori Enumeration

We design an Apriori enumeration method to efficiently discover all the valid patterns in a star partition. The principle of Apriori algorithm is to construct a lattice structure and enumerate all the possible candidate sets in a bottom-up manner. Its merit lies in the monotonic property such that if a candidate set is not valid, then all its supersets can be pruned. Thus, it works well in practice in spite of the exponential search space.

Our customized Apriori Enumerator is presented in Algorithm 3. Initially, the edges (pairs of objects) in the star constitute the bottom level (Lines 2-6) and invalid candidates are excluded (Line 4). An indicator *level* is used to control the object size for candidate set join. During each iteration (Lines 8-29), only candidates with object size equals to *level* are generated (Line 10). When two candidate sets c_1 and c_2 are joined, the new candidate becomes $c' = \langle c_1.O \cup c_2.O, c_1.T \cap c_2.T \rangle$ (Lines 11). To check the validity of the candidate, we calculate $\texttt{sim}(c'.T)$. If its simplified sequence is empty, c' is excluded from the next level (Line 12). This ensures that then all the candidates with $P.O \supseteq c'.O$ are pruned. If a candidate cannot generate any new candidate, then it is directly reported as a valid closed pattern (Lines 16-20). To further improve the performance, we adopt the idea of *forward closure* [18, 19] and aggressively check if the union of all the current candidates form a valid pattern (Lines 22-27). If yes, we can early terminate the algorithm and output the results.

Example 5. *As shown in Figure 5(c), in the bottom level of the lattice structure, candidate $\langle 3, 6 : 3 \rangle$ is pruned because its simplified sequence is empty. Thus, all the object sets containing $\langle 3, 6 \rangle$ can be pruned. The remaining two candidates (i.e., $\langle 3, 4 : 1, 2, 3 \rangle$ and $\langle 3, 5 : 2, 3 \rangle$) derive a new $\langle 3, 4, 5 : 2, 3 \rangle$ which is valid. By the forward closure checking, the algorithm can terminate and output $\langle 3, 4, 5 : 2, 3 \rangle$ as the final closed pattern.*

5.3 Put It Together

We summarize the workflow of SPARE in Figure 5 as follows. After the parallel clustering in each snapshot, for ease

Algorithm 3 Apriori Enumerator

Require: Sr_s
$C \leftarrow \emptyset$
for all edges $c = \langle o_i \cup o_j, T_{o_i} \cap T_{o_j} \rangle$ in Sr_s **do**
  if $\texttt{sim}(T_{o_i} \cap T_{o_j}) \neq \emptyset$ **then**
    $C \leftarrow C \cup \{c\}$
  end if
end for
level $\leftarrow 2$
while $C \neq \emptyset$ **do**
  for all $c_1 \in C$ **do**
    for all $c_2 \in C$ and $|c_2.O \cup c_2.O| =$ level **do**
      $c' \leftarrow \langle c_1.O \cup c_2.O : (c_1.T \cap c_2.T) \rangle$
      if $\texttt{sim}(c'.T) \neq \emptyset$ **then**
        $C' \leftarrow C' \cup \{c'\}$
      end if
    end for
    if no c' is added to C' **then**
      if c_1 is a valid pattern **then**
        output c_1
      end if
    end if
  end for
  $O_u \leftarrow$ union of $c.O$ in C
  $T_u \leftarrow$ intersection of $c.T$ in C
  if $\langle O_u, T_u \rangle$ is a valid pattern **then**
    output $\langle O_u, T_u \rangle$
    break;
  end if
  $C \leftarrow C'; C' \leftarrow \emptyset;$ level $\leftarrow$ level $+ 1$
end while
output C

of presentation, we used an aggregated graph G_A to capture the clustering relationship. However, in the implementation of the map phase, there is no need to create G_A in advance. Instead, we simply need to emit the edges within a star to the same reducer. Before sending stars to reducers, we use a simple best-fit strategy to calculate the star allocation. In best-fit strategy, the most costly unallocated star is assigned to the most empty reducers, where we use the edges in the star as a cost estimation. Each reducer is an Apriori Enumerator. When receiving a star Sr_i, the reducer creates initial candidate patterns. Specifically, for each $o \in Sr_i$, a candidate pattern $\langle o, i : e(o, i) \rangle$ is created. Then it enumerates all the valid patterns from the candidate patterns. The pseudocode of SPARE is presented in Algorithm 4.

Compared with TPMP, the SPARE framework does not rely on snapshot replication to guarantee correctness. In addition, we can show that the patterns derived from a star partition are unique and there would not be duplicate patterns mined from different star partitions.

Theorem 8 (Pattern Uniqueness)**.** *Let Sr_i and Sr_j $(i \neq j)$ be two star partitions. Let P_i (resp. P_j) be the patterns discovered from Sr_i (resp. Sr_j). Then, $\forall p_i \in P_i, \forall p_j \in P_j$, we have $p_i.O \neq p_j.O$.*

Proof. We prove by contradiction. Suppose there exist $p_i \in P_i$ and $p_j \in P_j$ with the same object set. Note that the center vertex of the star is associated with the minimum id. Let o_i and o_j be the center vertices of the two partitions

Algorithm 4 Star Partition and ApRiori Enumerator

Require: list of $\langle t, S_t \rangle$ pairs
—Map phase—
for all $C \in S_t$ **do**
  for all $o_1 \in C, o_2 \in C, o_1 < o_2$ **do**
    emit a $\langle o_1, o_2, \{t\} \rangle$ triplet
  end for
end for
—Partition and Shuffle phase—
for all $\langle o_1, o_2, \{t\} \rangle$ triplets **do**
  group-by o_1, emit $\langle o_1, Sr_{o_1} \rangle$
end for
—Reduce phase—
for all $\langle o, Sr_o \rangle$ **do**
  AprioriEnumerator(Sr_o)
end for

and we have $o_i = o_j$. However, P_i and P_j are different stars, meaning their center vertices are different (i.e., $o_i \neq o_j$), leading to a contradiction. □

Theorem 8 implies that no mining efforts are wasted in discovering redundant patterns in the SPARE framework, which is superior to the TRPM baseline. Finally, we prove the correctness of the SPARE framework.

Theorem 9. *The SPARE framework guarantees completeness and soundness.*

Proof. See Appendix A.2. □

6. EXPERIMENTAL STUDY

In this section, we evaluate the efficiency and scalability of our proposed distributed GCMP detectors on real trajectory datasets. All the experiments are carried out in a cluster with 12 nodes, each equipped with four quad-core 2.2GHz Intel processors, 32GB memory and gigabit Ethernet.

Environment Setup: We use Yarn[2] to manage our cluster. We pick one machine as the Yarn's master node, and the remainings reserve one core and 2GB memory for Yarn processes. We deployed our GCMP detector on Apache Spark 1.5.5 [20] with the remaining 11 nodes as the computing nodes. To fully utilize the computing resources, we configure each node to run five executors, each taking three cores and 5GB memory. In Spark, one of the 55 executor is taken as the Application Master for coordination, therefore our setting results in 54 executors. All our implementations as well as cluster setups are open-sourced in XXX.

Datasets: We use three real trajectory datasets in different application scenarios:

- Geolife [3]. The dataset essentially keeps all the travel records of 182 users for a period of over three years, including multiple kinds of transportation modes (walking, driving and taking public transportation). For each user, the GPS information is collected periodically and 91 percent of the trajectories are sampled every 1 to 5 seconds.

[2]http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html
[3]http://research.microsoft.com/en-us/projects/geolife/

- Shopping[4]. The dataset contains trajectories of visitors in the ATC shopping center in Osaka. To better capture the indoor activities, the visitor locations are sampled every half second, resulting in 13,183 long trajectories.
- Taxi. The dataset tracks the trajectories of 15,054 taxies in Singapore over August 2008. The sampling rate is around 30 seconds.

Preprocessing: We replace timestamps with global sequences (starting from 1) for each dataset. We set a fixed sampling rate for each dataset (i.e., Geolife = 5 seconds, Shopping=0.5 seconds, Taxi = 30 seconds) and use linear interpolation to fill missing values. For the clustering method, we use DBSCAN [21] and customize its two parameters ϵ and $minPt$ [5]. We set $\epsilon = 5$, $minPt = 10$ for GeoLife and Shopping datasets; and $\epsilon = 20$, $minPt = 10$ for Taxi dataset. Note that other clustering methods and settings can also be applied. The clustered snapshots are stored in HDFS as $\langle t, S_t \rangle$ pair, where t is the timestamp, S_t contains the clusters at snapshot t. After preprocessing, the statistics of the three datasets are presented in Table 5.

Attributes	Shopping	Geolife	SingTaxi
# objects	13,183	18,670	15,054
# data points	41,052,242	54,594,696	296,075,837
# snapshots	16,931	10,699	44,364
# clusters	211,403	206,704	536,804
avg. cluster size	171	223	484

Table 5: Statistics of data set

Parameters: To systematically study the performance of our algorithms, we conduct experiments on various parameter settings. The parameters to be evaluated are listed in Table 6, with default settings in bold.

Variables	Meaning	Values
M	min size of object set	5, 10, **15**, 20, 25
K	min duration	120, 150, **180**, 210, 240
L	min local duration	10, 20, **30**, 40,50
G	max gap	10, 15, **20**, 25, 30
O_r	ratio of objects	20%,40%, 60%, 80%, **100%**
T_r	ratio of snapshots	20%,40%, 60%, 80%, **100%**
N	number of machines	1, 3, 5, 7, 9, **11**

Table 6: Variables and their default values

6.1 Performance Evaluation

Varying M: Figures 7 (a),(e),(i) present the performance with increasing M. The SPARE framework demonstrates a clear superiority over TRPM framework, with a boosting factor of 2.7 times in Shopping, 3.1 times in Geolife and 7 times in Taxi. As M increases, the running time of both frameworks slightly improves because the number of clusters in each snapshot drops, generating fewer valid candidates.

Varying K: The performance with increasing K is shown in Figure 7 (b),(f),(j). SPARE tends to run faster, whereas the performance of TRPM degrades dramatically. This is caused by the *sequence simplification* procedure in SPARE, which can prune many candidates with large K. However, the line sweep algorithm in TRPM does not utilize such property for pruning. It takes longer time because more replication data has to be handled in each partition.

Varying L: Figures 7 (c)(g)(k) present the performances with varying L. When $L = 10$, SPARE can outperform TRPM by around 10 times. We also observe that there is a significant performance improvement for TPRM when L increases from 10 to 20 and later the running time drops smoothly. This is because η is proportional to $O(L + 1/L)$. When L is small (i.e., from 10 to 20), η decreases quickly (i.e., 30%). As L further increases (e.g, from 20 to 30), η decreases steadily (e.g, 15%).

Varying G: Figures 7 (d)(h)(l) present the performances with increasing G. TRPM is rather sensitive to G. When G is relaxed to larger values, more valid patterns would generate. TPRM has to set a higher replication factor and its running time degrades drastically when G increases from 20 to 30. In contrast, with much more effective pruning strategy, our SPARE scales very well with G.

Varying O_r: Figures 8 (a)(b)(c) present the performances wrt. increasing number of total objects. The x-axis are the ratio of objects sampled from the corresponding datasets. We can see that the performance of both TRPM and SPARE decreases linearly to the the number of total objects. This is reasonable since larger O_r results in more clusters per snapshot, which affects the line-sweep in TRPM and the aggregate graph in SPARE. However in all cases, SPARE consistently outperforms TRPM.

Varying T_r: Figures 8 (d)(e)(f) present the performances wrt. increasing number of total snapshots. The x-axis represents the size of snapshots sampled from the corresponding datasets. We again observe the linear decrease of the performances wrt. the number of snapshots. In all cases, SPARE is faster than TRPM. We further observe that as T_r grows, the gaps between SPARE and TRPM tends to be larger, this is because larger T_r results larger partition size, and more snapshots needs to be swept in TRPM.

6.2 Analysis of SPARE framework

In this part, we extensively evaluate the advantage brought by sequence simplification and load balancing techniques.

6.2.1 Power of sequence simplification

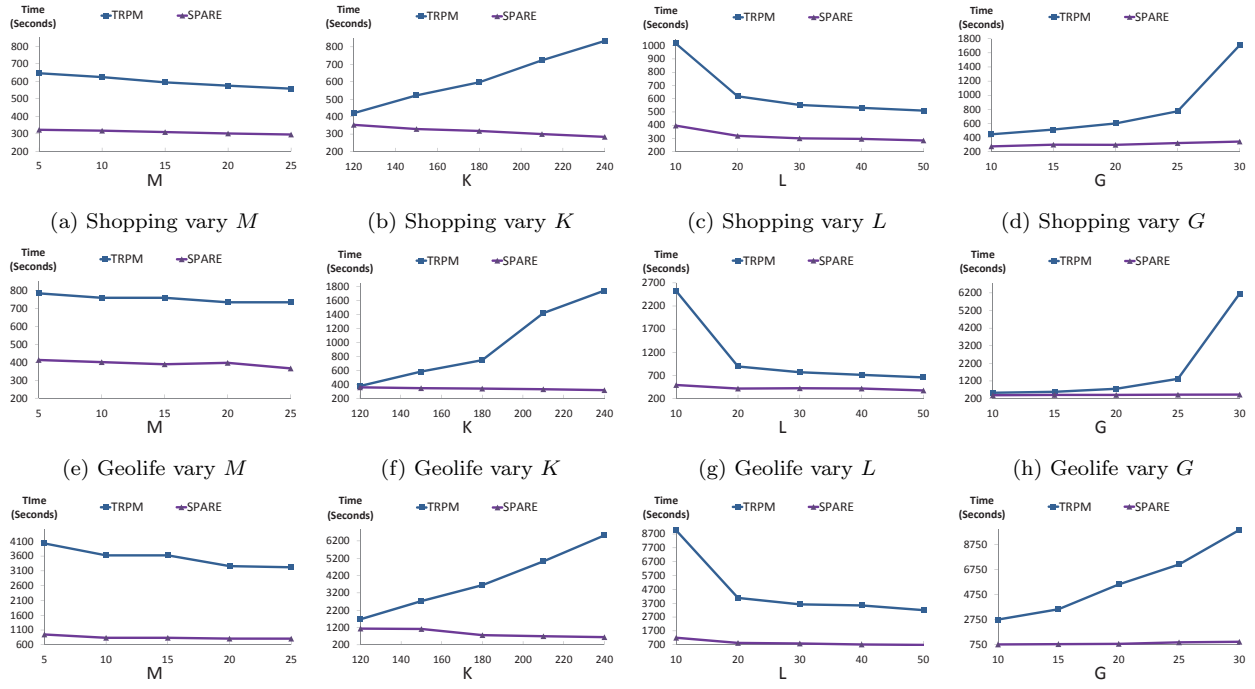
One of the core techniques used in SPARE is the *sequence simplification* (see Section 5.2.1). As shown in Algorithm 3, we use $\texttt{sim}(T)$ to prune false candidates. To study the power of sequence simplification, we collect the following two statistics: (1) the number of pairs that are shuffled to reducers and (2) the number of pairs that are fed to the Apirori enumeration. The difference of the two quantities the the pairs that are pruned by the sequence simplification. The statistics are shown in Table 7 under default parameters. The result states that the *sequence simplification* is a very powerful pruning technique. It cuts off near 90% of the initial pairs, which significantly reduces the costs of later enumerations. This confirms the necessity and usefulness of design such a technique.

6.2.2 Load Balance

To study why SPARE is more efficient, we analyze the running time of SPARE each map-reduce stage. To compare

[4]http://www.irc.atr.jp/crest2010_HRI/ATC_dataset/

[5]ϵ is the maximum neighborhood range to be concerned, $minPt$ is the minimum number of objects to form a dense region.



(a) Shopping vary M (b) Shopping vary K (c) Shopping vary L (d) Shopping vary G

(e) Geolife vary M (f) Geolife vary K (g) Geolife vary L (h) Geolife vary G

(i) Taxi vary M (j) Taxi vary K (k) Taxi vary L (l) Taxi vary G

Figure 7: Performance of SPARE and TRPM on real datasets under different pattern parameters.

Data Set	Shopping	Geolife	Taxi
Before pruning	878,309	1,134,228	2,210,101
After pruning	76,672	123,410	270,921
Prune ratio	91.2%	89.1%	87.7%

Table 7: Pruning power of SPARE

with, we use the SPARE-RD which is SPARE with random task allocation as a baseline. The detail anatomy is shown in Figure 9(a). We can see that on all three datasets, the reduce phase for both SPARE and SPARE-RD takes the majority time. The overall improvement of SPARE is around 10-13% under the default settings. We observe that the map and shuffle time of SPARE and SPARE-RD are identical, where SPARE spends a very small portion of extra time (4% of the total time) in applying the best-fit strategy. The time spent is worthwhile as in the reduce phase SPARE saves around 20% of the time.

We then look at the workload distribution of SPARE and SPARE-RD on real datasets. We collect the statistics from executors and report their reduce times in Figure 9. The figures show that SPARE is able to provide a more balanced task allocation in all three cases. Since SPARE-RD takes random assignment of stars, it is likely to assign many large stars to the same executor. In general cases, SPARE is recommended as it offers more efficiency while takes only a small extra time for planning.

6.3 Scalability and Comparison with Existing Solutions

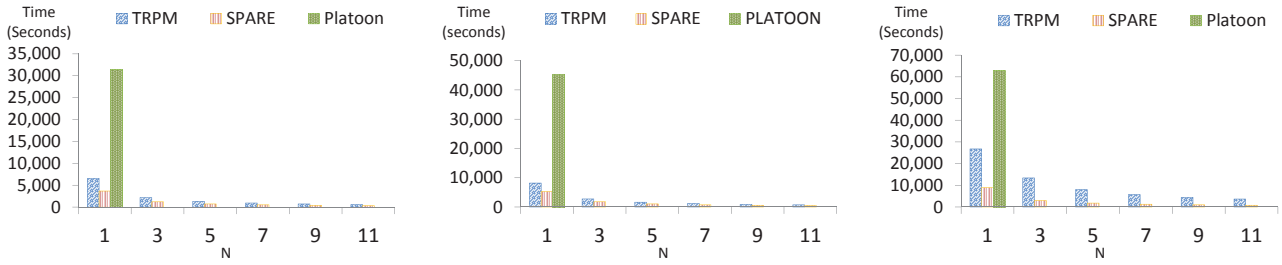
We then study the scalability of our schemes. Since existing solutions run on a single machine, it is interesting to compare their performances. We choose *platoon* as the comparing scheme as it is generic and more efficient than *swarm* and *group* The results are presented in Figure 10. There are two observations. First, the centralized scheme (e.g. platoon) are not suitable for discover patterns in large-scale trajectories. It takes near 16 hours for discovering *platoon*s in Taxi data. As a comparison, our TRPM and SPARE achieves 2.4 times and 7.1 times speedup. This is because even there is only 1 machine, our TRPM and SPARE is still able to leverage the 15 cores in the machine for parallelism. Second, we see that both TRPM and SPARE demonstrate good scalability wrt. N. As the number of machines increases from 1 to 11 (i.e., 11 times), both TRPM and SPARE achieves near 11 times speedup. Note that SPARE at most achieves 94 times speedup as compared to *platoon* using all 11 machines.

7. CONCLUSION

In this paper, we present our framework SPARE for detecting General Co-Movement Patterns (GCMP) over large-scale trajectories. Our GCMP is versatile in representing all existing co-movement patterns while being free from the *loose-connection* anomaly. Our SPARE framework, comprising a novel star-partition with several optimization techniques, has shown superior performances and scalability in three real datasets with hundreds of millions trajectories.

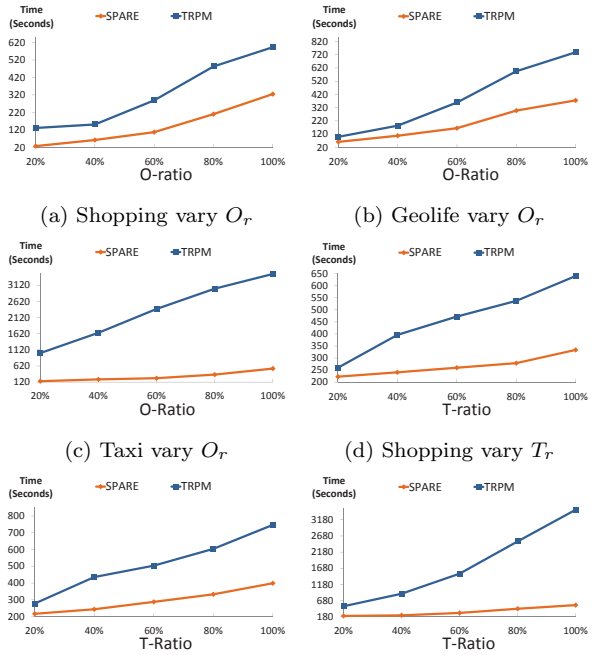
8. REFERENCES

[1] Y. Zheng, Y. Liu, J. Yuan, and X. Xie, "Urban computing with taxicabs," in *Proceedings of the 13th international conference on Ubiquitous computing*, pp. 89–98, ACM, 2011.



(a) Shopping vary N (b) Geolife vary N (c) Taxi vary N

Figure 10: Scalability of SPARE and TRPM wrt. N

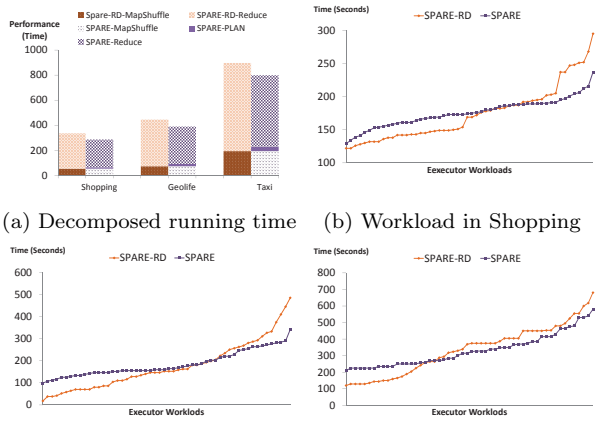


(a) Shopping vary O_r (b) Geolife vary O_r

(c) Taxi vary O_r (d) Shopping vary T_r

(e) Geolife vary T_r (f) Taxi vary T_r

Figure 8: Scalability of SPARE and TRPM wrt. O_r and T_r



(a) Decomposed running time (b) Workload in Shopping

(c) Workload in Geolife (d) Workload in Taxi

Figure 9: Load balance of SPARE and SPARE-LB

[2] Z. Li, B. Ding, J. Han, R. Kays, and P. Nye, "Mining periodic behaviors for moving objects," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1099–1108, ACM, 2010.

[3] J. Bao, Y. Zheng, D. Wilkie, and M. F. Mokbel, "A survey on recommendations in locationbased social networks. submitted to," *Geoinformatica*, 2013.

[4] X. Li, *Managing moving objects and their trajectories*. PhD thesis, National University of Singapore, 2013.

[5] J. Gudmundsson and M. van Kreveld, "Computing longest duration flocks in trajectory data," in *Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems*, pp. 35–42, ACM, 2006.

[6] Y. Wang, E.-P. Lim, and S.-Y. Hwang, "Efficient mining of group patterns from user movement data," *Data and Knowledge Engineering*, vol. 57, no. 3, pp. 240–282, 2006.

[7] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen, "Discovery of convoys in trajectory databases," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 1068–1080, 2008.

[8] Z. Li, B. Ding, J. Han, and R. Kays, "Swarm: Mining relaxed temporal moving object clusters," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 723–734, 2010.

[9] Y. Li, J. Bailey, and L. Kulik, "Efficient mining of platoon patterns in trajectory databases," *Data and Knowledge Engineering*, 2015.

[10] J. Gudmundsson, M. van Kreveld, and B. Speckmann, "Efficient detection of motion patterns in spatio-temporal data sets," in *Proceedings of the 12th annual ACM international workshop on Geographic information systems*, pp. 250–257, ACM, 2004.

[11] Y. Zheng, "Trajectory data mining: an overview," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 6, no. 3, p. 29, 2015.

[12] D. Birant and A. Kut, "St-dbscan: An algorithm for clustering spatial–temporal data," *Data and Knowledge Engineering*, vol. 60, no. 1, pp. 208–221, 2007.

[13] P. Laube, M. van Kreveld, and S. Imfeld, "Finding remodetecting relative motion patterns in geospatial

lifelines," in *Developments in spatial data handling*, pp. 201–215, Springer, 2005.

[14] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973. doi:10.3138/FM57-6770-U75U-7727.

[15] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, "Skewtune: mitigating skew in mapreduce applications," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pp. 25–36, ACM, 2012.

[16] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica, "Shark: Sql and rich analytics at scale," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of data*, pp. 13–24, ACM, 2013.

[17] E. Coppa and I. Finocchi, "On data skewness, stragglers, and mapreduce progress indicators," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, pp. 139–152, ACM, 2015.

[18] J. Wang, J. Han, and J. Pei, "Closet+: Searching for the best strategies for mining frequent closed itemsets," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 236–245, ACM, 2003.

[19] J. Pei, J. Han, R. Mao, *et al.*, "Closet: An efficient algorithm for mining frequent closed itemsets.," in *ACM SIGMOD workshop on research issues in data mining and knowledge discovery*, vol. 4, pp. 21–30, 2000.

[20] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pp. 2–2, USENIX Association, 2012.

[21] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 226–231, ACM, 1996.

APPENDIX

A. PROOFS OF THEOREMS

A.1 Proof of Theorem 2 and 3

Proof. We formalize optimization of Γ using a linear algebra model as follows: Let G_A be an aggregate graph, with a $n \times n$ adjacent matrix J. A vertex order is in fact a permutation of J. Therefore, the adjacent matrices of any reordered graphs can be represented as PJP^T where $P \in \mathbb{P}$ is a *permutation matrix* [6] with dimension n. Since in star partition, we assign each edge $e(i,j)$ in G_A to the lower vertex, then the matrix $B = \text{triu}(PJP^T)$ [7] represents the assignment matrix wrt. P (i.e., $b_{i,j} = 1$ if vertex j is in star Sr_i). Let vector $\vec{b}$ be the *one*[8] vector of size n. Let $\vec{c} = B\vec{b}$, then each c_i denotes the number of edges in star Sr_i. Thus, Γ can be represented as the infinity norm of $B\vec{b}$. Let Γ^* be the minimum Γ among all vertex orders. Γ^* can then be formulated as follows:

$$\Gamma^* = \min_{P \in \mathbb{P}} ||B\vec{b}||_\infty \text{ ,where } ||B\vec{b}||_\infty = \max_{1 \le j \le n} (c_j) \qquad (1)$$

Let $B*$ be the assignment matrix wrt the optimal vertex order. Since we have a star for each object, by the degree-sum formula and pigeon-hole theorem, $\Gamma^* = ||B^*\vec{b}||_\infty \ge d/2$. Next, given a numbering P, let $e_{i,j}$ be an entry in PAP^T. Since edges in graph G are independent, then $e_{i,j}$s are independent. Moreover, let d_i denote the degree of vertex i, then $E[d_i] = E[\Sigma_{1 \le j \le n} e_{i,j}] = d$. This is because renumbering the vertexes does not affect the average degree. Since $B = \text{triu}(PAP^T)$, entries in B can be written as :

$$b_{i,j} = \begin{cases} e_{i,j}, i > j \\ 0, otherwise \end{cases}$$

There are two observations made. First, since $e_{i,j}$s are independent, $b_{i,j}$s are independent. Second, since $i > j$ and $e_{i,j}$s are independent. $E[b_{i,j}] = E[e_{i,j}]E[i > j] = E[e_{i,j}]/2$.

By definition, $c_i = \Sigma_{1 \le j \le n} b_{i,j}$, is a sum of n independent 0-1 variables. Taking expectation on both sides, we get: $E[c_i] = E[\Sigma_{1 \le j \le n} b_{i,j}] = E[\Sigma_{1 \le j \le n} e_{i,j}]/2 = d/2$. Let $\mu = E[c_i] = d/2$, $t = \sqrt{n \log n}$, by Hoeffding's Inequality, the following holds:

$$\begin{aligned} Pr(c_i \ge \mu + t) &\le \exp(\frac{-2t^2}{n}) \\ &= \exp(-2 \log n) = n^{-2} \end{aligned}$$

The first step is due to the fact that all $b_{i,j}$ are bounded in the range of [0,1]. Next, since the event $(\max_{1 \le j \le n}(c_j) \ge \mu + t)$ can be viewed as $\cup_{c_i}(c_i \ge \mu + t)$, by Union Bound, we achieve the following:

$$\begin{aligned} Pr(\Gamma \ge \mu + t) &= Pr(\max_{1 \le j \le n} (c_j) \ge \mu + t) \\ &= Pr(\cup_{c_i}(c_i \ge \mu + t)) \\ &\le \Sigma_{1 \le i \le n} Pr(c_i \ge \mu + t) \\ &= n^{-1} = 1/n \end{aligned}$$

Substitute back t and μ, we achieve the following concise form:

$$Pr(\Gamma \ge (d/2 + \sqrt{n \log n})) \le 1/n$$

This indicates that, the probability of $(\Gamma - d/2)$ being less than or equal to $O(\sqrt{n \log n})$ is $(1 - 1/n)$. With the fact that $\Gamma^* \ge d/2$, we conclude that with probability greater than $(1 - 1/n)$, the difference between Γ and Γ^* is less than $O(\sqrt{n \log n})$. When the aggregated graph is *dense* (i.e., $d \ge \sqrt{12 \log n}$), we may use the Chernoff Bound instead of Hoeffding's Inequality to derive a tighter bound of $O(\sqrt{\log n})$ with the similar reasoning. □

A.2 Proof of Theorem 9

[6] an identity matrix with rows shuffled

[7] triu is the upper triangle part of a matrix

[8] every element in $\vec{b}$ is 1

Proof. For soundness, let P be a pattern enumerated by SPARE. Since the enumerate algorithm only reduces the time sequences, for any two objects $o_1, o_2 \in P.O$, the edge $e(o_1, o_2)$ is a superset of $P.T$. By the definition of star, $\forall t \in T, C_t(o_1) = C_t(o_2)$. As T is a valid sequence, by the definition of GCMP, P is a true pattern. For completeness, let P is a true pattern. Let s be the object with smallest ID in $P.O$. We prove that P must be output by Algorithm 3 with input Sr_s. First, based on the definition of star, every object in $P.O$ appears in Sr_s. Since $P.T$ is decomposable, then $\forall O' \subseteq O$, the time sequence of O' is a super set of a decomposable sequence. This indicates that O' would not be eliminated by any `sim` operations in Algorithm 3. Next, we prove at every iteration $level \leq |P.O|$, $P.O \subset O_u$, where O_u is the forward closure. We prove by induction. When $level = 2$, it obviously holds. If $P.O \subset O_u$ at $level{=}i$, then any subset of $P.O$ with size i are in the candidate set. In $level$ $i + 1$, these subsets are able to grow to a bigger subset (in last iteration, they grow to $P.O$). This suggests that no subsets are removed by Lines 16-30. Then, $P.O \subset U_{i+1}$ holds. In summary, $P.O$ does not pruned by simplification, monotonicity and forward closure, therefore P must be returned by SPARE.

□